Recommendations for implementing a Bitcoin wallet using smart card

Ricardo V. Fritsche, Lucas M. Palma, Jean E. Martina

Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
Campus Universitário Trindade Cx.P. 476 / CEP 88040 – Florianópolis – SC – Brazil

ricardo@grad.ufsc.br, lucas.palma@posgrad.ufsc.br, jean.martina@.ufsc.br

Abstract. *On the Bitcoin peer-to-peer electronic cash system the user's funds are protected by private keys that must be kept safe. In this work we made a review on cryptography, the Bitcoin protocol and secure elements, then we dived into the project of hardware wallets, discussing different requirements and ways to construct one. Our proposed device uses an anti-tamper Java Card to store the private keys. We considered variations of the device, one with a dedicated touchscreen and another with NFC to integrate with a mobile phone. We analyzed security aspects of the project, made recommendations and described some challenges. Finally, we implemented our own open source prototype, showing the architecture of the project, its components, the requirements, the APDU communication protocol and the results.*

1. Introduction

Bitcoin is a peer-to-peer electronic cash system that allows any two willing parties to transact directly with each other without the need for a trusted third party [1]. Bitcoin has grown rapidly, giving birth to a novel industry, involving other cryptocurrencies, blockchain technology and new economic dynamics [2]. Many users might prefer to leave their coins on an exchange as it seems to be easier and works almost like the current online banking solution [3]. The problem is that if the users are not the real owners of their funds, the whole idea of decentralization is lost, giving space to government censorship, retention of user's funds and other threats [7].

The viable alternative for the users is to use hardware wallets, devices that offers the best balance between very high security and ease of use [23]. The main wallets on the market have limitations on the number of cryptocurrencies supported and a high price, but the main concern is that they still present some security flaws, like the exposure of the communication channel [6] and the openness to a Supply Chain and Evil Maid attacks [21].

In this academic work we did a thorough description of some important security aspects of managing keys and Bitcoin addresses. Not only that, we developed an open source prototype that might be a starting point for more robust wallets that are low cost, secure and can be adapted for different coins. We analyzed different hardware wallet projects possibilities, with and without a dedicated screen. We explore the solutions to encrypt the communication channel, preventing Man-in-the-Middle (MitM) attacks [6]

by using password-authenticated secure channel protocol (SRP) [12] and public key cryptography.

2. Fundamentals

In this chapter we make a brief explanation of the main cryptography concepts used by the Bitcoin protocol. We also detail the protocol itself, explore Java Cards and related works.

2.1. Cryptography

Cryptographic hash function takes a variable-length block of data as input and returns a fixed-size value called hash [13]. Any change in the message will have a very high probability to produce a distinct hash, allowing to check for data integrity and various others applications, notably digital signatures [14]. They present different properties: deterministic, fast to compute, preimage resistant, second preimage resistant and strong collision resistant [13]. The Secure Hash Algorithm (SHA) is a family of widely used hash functions that is present in the Bitcoin protocol.

Message Authentication Code (MAC) algorithm has two purposes: to verify the integrity and the authenticity of a message [13]. To form a MAC one can use a cryptographic hash function, such as SHA-256, combined with a secret key, this is known as Hash-based Message Authentication Code (HMAC) [13]. They also can be used to generate pseudorandom numbers of fixed length, which is explored by the Bitcoin protocol to generate wallets.

Key Derivation Function (KDF) takes an input (usually a password or passphrase) and produces a secret key that can fulfil some required format, for example, a certain length and higher entropy. More than that, it is used to prevent brute force attacks, as two elements are present in the function: the use of a salt, avoiding the pre-computation of keys and the usage of iteration, requiring more computational power to execute the function [16].

Digital Signatures are one of most important instances of public-key cryptography [14], with applications on a wide range of areas, including secure e-commerce, legal signing of contracts, secure software updates, online banking and Bitcoin, where a transaction is securely signed, allowing only the owner of the signing private key to spend funds associated with a referenced public key [5]. They present three important aspects: authentication, integrity and non-repudiation. It should be computationally infeasible to generate a valid signature for any given message without knowing the private key, while verifying the signature using the public key should be trivial.

Elliptic Curve Digital Signature Algorithm (ECDSA) is used as an alternative to RSA/DSA and other schemes for digitally signing messages, with the advantage that the same level of security can be achieved using a much smaller key size [13]. Bitcoin protocol has chosen the secp256k1 curve parameters defined in the Standards for Efficient Cryptography (SEC) to make all the ECDSA operations [17].

2.2. Bitcoin

Bitcoin is a Peer-to-Peer Electronic Cash System initially developed by Satoshi Nakamoto in late 2008 to allow any two willing parties to transact directly with each other without the need for a trusted third party [1]. All Bitcoin transactions are transmitted to the network to be propagated to all participating nodes forming a distributed database called blockchain. To prevent double spending of coins and reach global decentralized consensus, the nodes on the network execute a proof-of-work algorithm based on cryptography hashing. To transfer bitcoins to a recipient, one must own the ECDSA's private key that will sign the transaction and release the funds. The transactions are designed to be cheap and fast (disregarding the whole energetic cost to mine coins). Bitcoin uses addresses as pseudonyms, where the addresses are not tied to a person or company, but to a cryptography public key. When miners execute the proof-of-work algorithm before all others nodes, they are rewarded with freshly created bitcoins and the fees from the transactions. There is a limitation of total number of coins on the specification of the protocol, only 21 million bitcoins can be created, making Bitcoin deflationary [18].

2.3. Bitcoin wallets

Bitcoin wallets generally refers to the client software used to manage bitcoin private keys, generate addresses, forge transactions and aggregate balance information from the Bitcoin network. Wallets comes in different formats, ranging from simple local storage of keys on a desktop computer, to mobile applications, dedicated hardware devices, paper wallets or even brain wallets [4].

The first generation of Bitcoin wallets, like the *Bitcoin Core Wallet*, generates random keys and store them on the local filesystem, encrypted by a password or not [4, 18]. This approach presents headaches and security exposure. The user has to constantly backup the keys to other devices and trust that the computer is not infected by any kind of malware and that no unauthorized person has access to it. Hierarchical Deterministic Wallets (HD wallets) [15], on the other hand, represent a better solution. In this kind of wallet, any number of keys can be derived from a single master key, called seed, forming a hierarchic tree of accounts, each with its public and private keys. To be able to backup the wallet, the seed phrase is generated from a random number that is converted to a phrase with 12, 15, 18, 21 or 24 words from a standard wordlist, allowing different wallet software to generate compatible seed phrases. With 12 words the generated entropy is 128 bits and with 24 words the entropy is 256 bits. The order of the words matter and the phrase contain a checksum [24]. To make a HD wallet from a seed phrase, the words are used to derive a longer seed through the use of the key-stretching function PBKDF2 with HMAC-SHA512 [18].

2.4. Java Card

Secure elements (SE) are hardware parts designed to be tamper resistant, not allowing an attacker to retrieve or modify information in the its secure memory [20]. They are resilient to microprobing, software attacks, eavesdropping and fault generation [11].

Java Card is a SE platform released in 1996 to simplify the development of software for smart cards by offering portability and security [8, 9]. Java Card products are based on the Java Card Platform specifications, which allows a Java Card applet to be written once and run on different smart cards running a virtual machine on top of its operational system.

2.5. Related work

Gkaniatsou A., Arapinis M. and Kiayias A. released a paper in 2017 [6] where they reverse-engineer the APDU communication protocol of the *Ledger Nano S* hardware wallet [19], mounting a series of attacks to demonstrate the vulnerabilities. The attacks are really serious and serve as a reminder of the risks of sending clear text data through an insecure channel.

With that in mind we investigated different possibilities to deploy a secure channel on a Java Card and found the work of Hölzl, M., et al. (2015) [12]. The authors design, implement and evaluate the use of the password-authenticated secure channel protocol (SRP).

Beyond our informal research on current wallets products, we selected an academic paper by Eskandari S., et al. [4] where the main wallets solutions until the year of 2015 are mapped, studied and compared, mainly in the point of view of usability and user experience. Table 1 show our proposal as an additional last line to allow comparison with the others categories. As the keys are stored on a secure element, which cannot be tampered, we gave Malware Resistant a full point (•). On the Resistant to Physical Theft aspect, we gave our solution the half point (◦) as the device can be stolen but no keys can be extracted. Finally, on the No New User Software we also gave our solution the half point (◦), considering that the user will access the wallet through a web interface, but it is necessary to install the *Connector* (described in the Development chapter) software or at least an extension to the web browser.

Table 1: A comparison of key management techniques for Bitcoin. Modified from Eskandari S., et al. [4]

Category	*Example*	Malware Resistant	Key(s) Kept Offline	No Trusted Third Party	Resistant to Physical Theft	Resistant to Physical Observation	Resilient to Password Loss	Resilient to Key Churn	Immediate Access to Funds	No New User Software	Cross-device Portability
Keys in Local Storage	Bitcoin Core			•		•	•	•	•		
Password-protected Wallets	MultiBit		◦	•	◦	•		•	•		
Offline Storage	Bitaddress	◦	•	•			•				•
Air-gapped Storage	Armory	◦	•	•		•	•	•			
Password-derived Keys	Brainwallet		•	•	◦			•	•	•	•
Hosted Wallet (Hot)	`Coinbase.com`						•	•	•	•	•
Hosted Wallet (Cold)		◦	•				•	•		•	•
Hosted Wallet (Hybrid)	Blockchain.info		◦	◦			•	•	•	•	•
Cash		•	•	•		•	•	•	•	•	•
Online Banking							•	•	•	•	•
Our solution		•	•	•	◦	•	•	•	•	◦	•

3. Development

In this chapter we explore different possibilities to build a hardware wallet, discuss relevant aspects considerations and present our open source prototype.

3.1. Different possibilities for the hardware wallet

A hardware wallet can be constructed in many different ways, bringing unique security advantages, implementation challenges and costs of production for each version. As we have seen in the first chapter, it is really important that the private keys are kept on a secure element, making the smart card (Java Card) the basic requirement in our analysis. Another general precondition is the communication with an external computer (desktop, notebook or even a mobile phone), so the wallet can interact with the Bitcoin network and the user.

The most basic version does not include a screen, although it has the lowest cost. To make it more secure, we must include a screen. For this we need a microcontroller that will be able to make the interconnection between the smart card, the screen, the optional buttons and the communication channels (like USB, Bluetooth, etc.). This will increase the cost of the device and make the development more complex, but it is an unavoidable requirement, as we cannot trust a third-party device.

3.2. Security considerations

Independent of the different constructed wallet, there are common security considerations that must be taken care of:

1. Protect the communication channel using SRP or other method;
2. Protect the device with a PIN using the Java Card PIN SDK;
3. Allow to enter the PIN on an untrusted device without revealing it by showing the keyboard numbers in a random fashion on the device;
4. Allow the personalization of the device to prevent *Evil Maid* attacks;
5. Make it possible to use plausible deniability by allowing any PIN to generate a valid wallet;
6. Protect the hardware against attacks by disabling debug ports, monitoring the components actively and passively and potting of device;
7. Be able to verify the genuineness of the SE by recording its unique public key on the Bitcoin blockchain.

3.3. Prototype

To go beyond the theoretical knowledge, we developed a functional prototype using a real smart card (Java Card). The prototype did not have a dedicated screen or buttons and did not use a secure channel on its communication, although it checks for the attestation of the genuineness of the smart card. Figure 1 shows an overview of the general architecture of the project.

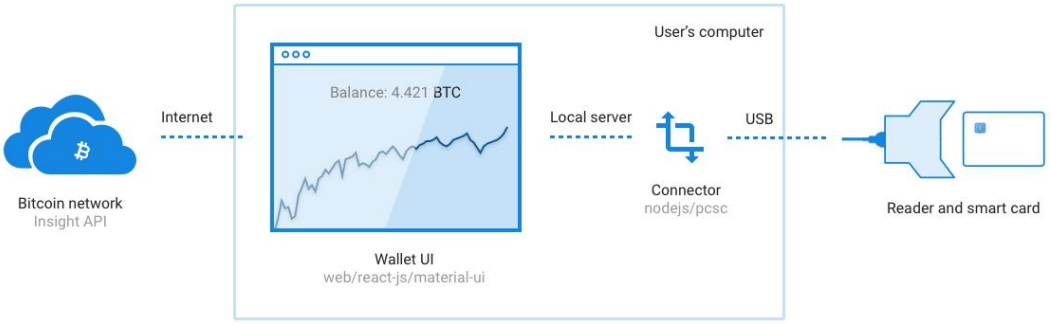

Figure 1: Overview of the architecture of the project.

The secure element selected was the NXP J3D081 EV1 smart card. It is a contact and contactless interface Java Card 3.0.1 running on Global Platform 2.2.1 with 80K of EEPROM. All the requirements (Figure 2) were implemented and tested with unit and integration tests. All the APDU commands and responses were documented.

SETUP

1.1. To define the PIN to access the device;
1.2. To change the PIN; (PIN required)
1.3. To generate the initial master private key of the device using a TRNG and the seed words for recovery (BIP-39)[1]; (PIN required)
1.4. To recover a wallet by defining the master private key using the seed words; (PIN required)
1.5. To verify that the seed words generates the master private key stored on the device; (PIN required)
1.6. To generate the attestation of genuineness keys using a TRNG.

ADDRESS GENERATION

2.1. To generate deterministic BIP-32 address (public/private keys). (PIN required)

SIGN TRANSACTIONS

3.1. To sign Bitcoin transactions using the keys of a BIP-32 path; (PIN required)

INTERACTION

4.1. To validate the user PIN;
4.2. To take actions based on PIN policies, like to erase the device if the PIN is wrong for 5 consecutive times;
4.3. To sign a challenge (SHA-256 hash) with the genuineness private key;
4.4. To respond with the genuineness public key of the device;
4.5. To erase the device if request by the user. (PIN required)

Figure 2: Requirements of the hardware device.

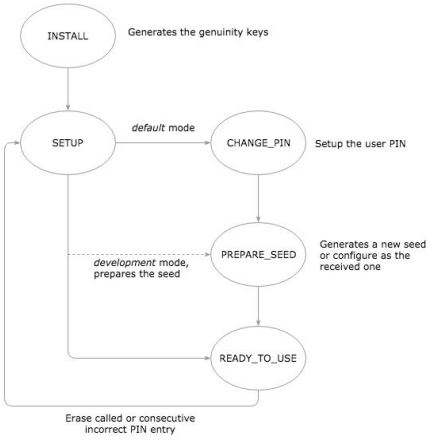

Figure 3: Possible states of the Applet.

We developed a middleware called *Connector* that makes the bridge between the USB smart card reader and the browser, by making it run a local web server on the client computer. This component starts a local HTTP server on port 28281 and listens to calls made from whitelisted domains or localhost. Java 1.8 was used for listening and responding to HTTP requests.

Figure 4 shows the main actions that the interface should support in a workflow format. These actions were used to draw and then implement the web interface (see Figure 5) as a Single Page Application (SPA). To communicate with the Bitcoin blockchain we used a widely adopted open source solution called *Insight-API*, maintained by *BitPay* [22].

The source code of this project is publicly available at GitHub on the following repository: *https://github.com/ricardovf/knox-wallet*. Be advised that this project is only a prototype and isn't ready to be used in a real product.

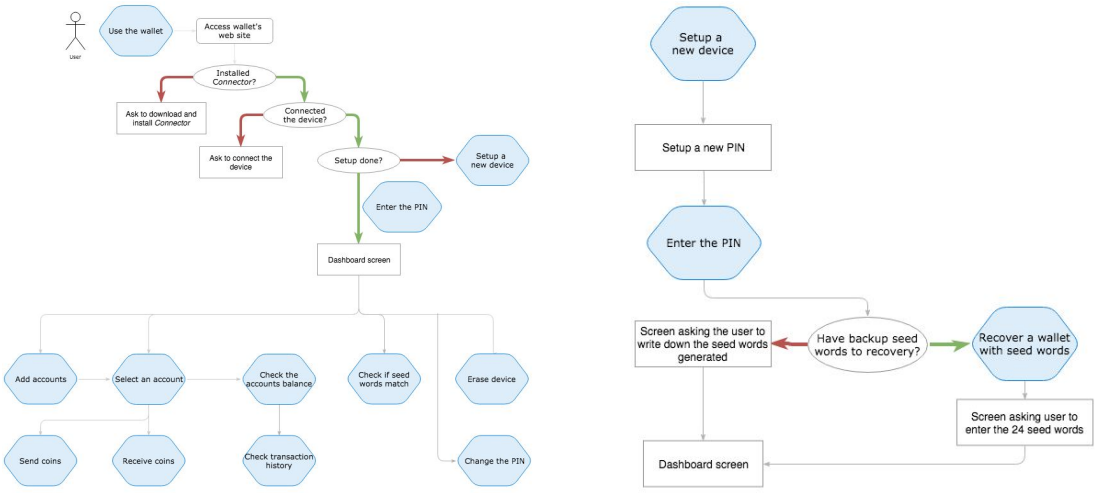

Figure 4: Workflow of navigation and setup on the wallet user interface.

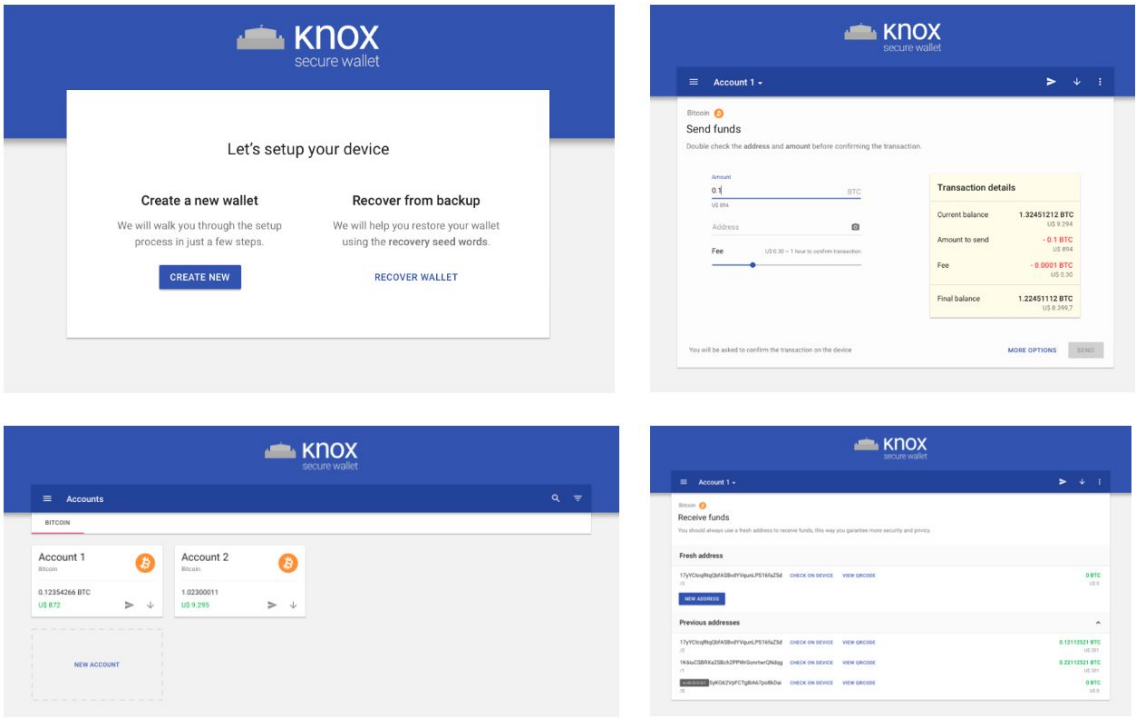

Figure 5: Setup, send funds, dashboard and receive funds screens.

4. Conclusion

We dived into the project of hardware wallets, discussing different requirements and ways to construct a device that uses an anti-tamper Java Card to store the private keys. We considered variations of the device and analyzed the security aspects of the project, making recommendations and describing some challenges. Finally, we implemented our own open source prototype, showing the architecture of the project, its components, the requirements, the APDU communication protocol and the results. The prototype constructed is limited in contrast to the more complete versions discussed, with a dedicated screen and an encrypted communication channel, but it can be further

developed to accommodate the full requirements and be deployed into the world as a complete product.

Hardware wallets projects are complex, involving integrated circuits and software that must communicate and integrate securely, considering different attack vectors, without letting out the importance of the final cost and usability. Nevertheless, we must constantly encourage the enhancement of the current solutions and develop novel approaches, making cryptocurrencies a practical everyday solution to contrast with the traditional financial system.

We believe there is a rich set of possibilities for future work related to this project, most of them on the practical side of extending and validating the security of the wallet:

1. Build prototype of the wallet with an integrated touchscreen;
2. Build a prototype using a contactless smart card (NFC) and a mobile phone as the device screen;
3. Systematically attack the wallet to find vulnerabilities and propose corrections;
4. Run usability test with real users to improve the wallet friendliness;
5. Implement the support for multiple cryptocurrencies, including ones not directly derived from Bitcoin Core;
6. Implement the support for multisignature scheme;
7. Find solutions to securely guarantee the genuineness of the integrated microcontroller;
8. To write an updated book on Java Card development utilizing open source libraries and tools, as the materials on the topic are very fragmented and diffuse.

References

[1] NAKAMOTO, Satoshi. **Bitcoin: A peer-to-peer electronic cash system**. 2008.
[2] Monopoly without a Monopolist: An Economic Analysis of the Bitcoin Payment System
[3] CONTI, M., E, S.K., LAL, C. & RUJ, S. 2018. **A Survey on Security and Privacy Issues of Bitcoin**. IEEE Communications Surveys & Tutorials :1–1.
[4] ESKANDARI, S., BARRERA, D., STOBERT, E. & CLARK, J. 2015. **A First Look at the Usability of Bitcoin Key Management**. Proceedings 2015 Workshop on Usable Security.
[5] COURTOIS, Nicolas T.; VALSORDA, Filippo; EMIRDAG, Pinar. **Private Key Recovery Combination Attacks: On Extreme Fragility of Popular Bitcoin Key Management, Wallet and Cold Storage Solutions in Presence of Poor RNG Events**. 2014.
[6] GKANIATSOU, A., ARAPINIS, M. & KIAYIAS, A. 2017. **Low-Level Attacks in Bitcoin Wallets**. Information Security :233–253.
[7] MAY, Timothy. **The Crypto Anarchist Manifesto**. Available at: <https://www.activism.net/cypherpunk/crypto-anarchy.html>. Accessed on may 29, 2018.
[8] BARBU, Guillaume. **On the security of Java Card platforms against hardware attacks**. 2012. Tese de Doutorado. Telecom ParisTech.
[9] HANSMANN, Uwe et al. **Smart card application development using Java**. Springer Science & Business Media, 2012.
[10] ATKINS, Steve. **JCF celebrates 15 year anniversary in Singapore – 10 billion Java Cards deployed worldwide**. Available at:

<https://contactlessintelligence.com/2012/09/21/jcf-celebrates-15-year-anniversary-in-singapore-10-billion-java-cards-deployed-worldwide/>. Accessed on may 29, 2018.
[11] KÖMMERLING, Oliver; KUHN, Markus G. **Design Principles for Tamper-Resistant Smartcard Processors**. Smartcard, v. 99, p. 9-20, 1999.
[12] HÖLZL, Michael et al. A password-authenticated secure channel for App to Java Card applet communication. **International Journal of Pervasive Computing and Communications**, v. 11, n. 4, p. 374-397, 2015.
[13] WILLIAM, Stallings. **Cryptography and network security: principles and practice**. Prentice-Hall, Inc, p. 23-50, 1999.
[14] PAAR, Christof; PELZL, Jan. **Understanding cryptography: a textbook for students and practitioners**. Springer Science & Business Media, 2009.
[15] GITHUB. **BIP32**. Available at: <https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki>. Accessed on jun 6, 2018
[16] KALISKI, Burt. **PKCS# 5: Password-based cryptography specification version 2.0**. 2000.
[17] CERTICOM, Standards for Efficient Cryptography. **SEC 2: Recommended Elliptic Curve Domain Parameters**. Version 2.0. 2010.
[18] ANTONOPOULOS, Andreas M. **Mastering Bitcoin: Programming the open blockchain**. O'Reilly Media, Inc. 2017.
[19] LEDGER. **Ledger Nano S**. Available at: <https://www.ledger.com/products/ledger-nano-s>. Accessed on oct 10, 2018.
[20] WIKIPEDIA. **Tamper resistance**. Available at: <https://en.wikipedia.org/wiki/Tamper_resistance>. Accessed on oct 10, 2018.
[21] RASHID, Saleem. **Breaking the Ledger Security Model**. Available at: <https://saleemrashid.com/2018/03/20/breaking-ledger-security-model/>. Accessed on oct 10, 2018.
[22] GITHUB. **The bitcoin blockchain API powering Insight**. Available at: <https://github.com/bitpay/insight-api>. Accessed on oct 10, 2018.
[23] BITCOIN. **Securing your wallet**. Available at: <https://bitcoin.org/en/secure-your-wallet>. Accessed on may 29, 2018.
[24] GITHUB. **BIP39**. Available at: <https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki>. Accessed on jun 6, 2018